5차시 과제 풀이

문제 목록

문제 번호	문제 이름	출처
BOJ11047	동전 0	
BOJ 11399	ATM	
BOJ 19941	햄버거 분배	KOI 2020 1차 고등부
BOJ 1931	회의실 배정	
BOJ 13975	파일 합치기 3	
BOJ 1916	최소비용 구하기	
BOJ 1197	최소 스패닝 트리	
BOJ 11877	홍수	COCI 2015/2016 #5
BOJ 14464	소가 길을 건너간 이유 4	USACO 2017 February Silver
BOJ 14908	구두 수선공	
BOJ 2878	캔디캔디	COCI 2020/2021 #1
BOJ 2180	소방서의 고민	
BOJ 18921	Cost Of Subtree	
BOJ 21761	초직사각형	KOI 2021 1차 고등부
BOJ 1422	숫자의 신	
BOJ 21740	도도의 수학놀이	
BOJ 12776	Swap Space	2016 ICPC World Finals
BOJ 5530	JOIOI 탑	JOI 2013
BOJ 18596	Monster Hunter	Ptz Camp Summer 2019 Day1

BOJ 11047 동전 0

강의 슬라이드 참고

```
#include <bits/stdc++.h>
using namespace std;

int N, K, A[11], R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
```

```
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=N; i>=1; i--){
        R += K / A[i];
        K %= A[i];
    }
    cout << R;
}
```

BOJ 11399 ATM

강의 슬라이드 참고

```
#include <bits/stdc++.h>
using namespace std;

int N, A[1010], R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    sort(A+1, A+N+1);

    for(int i=1,s=0; i<=N; i++){
        s += A[i];
        R += s;
    }
    cout << R;
}
```

BOJ 19941 햄버거 분배

사람들을 왼쪽에서 오른쪽으로 차례대로 보면서, 먹을 수 있는 가장 왼쪽 햄버거를 먹는 그리디가 성립한다.

$N \cdot K$가 충분히 작기 때문에 이분 매칭을 이용해 문제를 풀 수도 있다.

```
#include <bits/stdc++.h>
using namespace std;

int N, K, Eat[20202], R;
char A[20202];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> A[i];

    for(int i=1; i<=N; i++){
        if(A[i] == 'H') continue;
        for(int j=max(1, i-K); j<=min(N, i+K); j++){
            if(A[j] == 'H' && !Eat[j]){
                R++; Eat[j] = 1;
                break;
            }
        }
```

```
    }
    cout << R;
}
```

BOJ 1931 회의실 배정

강의 슬라이드 참고

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;

int N;
PII A[101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    sort(A+1, A+N+1, [](PII p1, PII p2){
        if(p1.y != p2.y) return p1.y < p2.y;
        return p1.x < p2.x;
    });

    int mx = 0, cnt = 0;
    for(int i=1; i<=N; i++){
        if(mx <= A[i].x) cnt++, mx = A[i].y;
    }
    cout << cnt;
}
```

BOJ 13975 파일 합치기 3

크기가 가장 작은 원소 2개를 합치는 그리디가 성립한다. 증명이 궁금하면 허프만 코딩을 공부해보자.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, A[1010101];

void Solve(){
    priority_queue<ll, vector<ll>, greater<>> pq;
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i], pq.push(A[i]);

    ll res = 0;
    while(pq.size() > 1){
        ll u = pq.top(); pq.pop();
        ll v = pq.top(); pq.pop();
        res += u + v;
        pq.push(u + v);
    }
    cout << res << "\n";
```

```
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int T; cin >> T;
    for(int tc=1; tc<=T; tc++) Solve();
}
```

BOJ 1916 최소비용 구하기

다익스트라 알고리즘을 구현하는 문제다.
여름방학 수업에서 다룰 예정인데, 혹시 궁금하다면 미리 공부해보는 것도 좋은 선택이다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using PLL = pair<ll, ll>;

int N, M, S, T, D[1010];
vector<PLL> G[1010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++){
        int s, e, x; cin >> s >> e >> x;
        G[s].emplace_back(e, x);
    }
    cin >> S >> T;

    priority_queue<PLL, vector<PLL>, greater<>> pq;
    memset(D, 0x3f, sizeof D);
    pq.emplace(0, S); D[S] = 0;
    while(pq.size()){
        auto [cst,v] = pq.top(); pq.pop();
        if(cst > D[v]) continue;
        for(auto i : G[v]){
            if(D[i.x] > cst + i.y) pq.emplace(D[i.x] = cst + i.y, i.x);
        }
    }
    cout << D[T];
}
```

BOJ 1197 최소 스패닝 트리

강의 슬라이드 참고

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

struct Edge{
    ll s, e, x;
    bool operator < (const Edge &edge) const { return x < edge.x; }
```

```
};

int N, M, P[10101];
vector<Edge> edges;

int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Merge(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return false;
    P[u] = v; return true;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M; edges.resize(M);
    for(auto &edge : edges) cin >> edge.s >> edge.e >> edge.x;
    sort(edges.begin(), edges.end());

    iota(P+1, P+N+1, 1);
    ll ans = 0;
    for(auto edge : edges) if(Merge(edge.s, edge.e)) ans += edge.x;
    cout << ans;
}
```

BOJ 11877 홍수

만들 수 있는 가장 큰 그릇의 크기는 $(N-2)(N-1)/2$이다. 만약 $X > (N-2)(N-1)/2$이면 무조건 불가능하다.

그릇의 양 끝 경계는 $N, N-1$로 하고, N과 $N-1$ 사이에 막대기를 적절히 넣어서 크기가 K가 되도록 만들면 된다. 이때 그릇을 만드는데 사용하지 않는 막대기들은 바깥쪽에 **그릇이 생성되지 않도록** 잘 배치하면 된다.

크기가 K인 그릇을 만드는 것은 길이가 작은 막대기부터 차례대로 보면서, 해당 막대기를 그릇 내부에 배치했을 때 그릇의 크기가 K를 넘어가지 않는다면 그릇에 넣는 그리디 전략이 성립한다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, X, K;
bool Use[1010101];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> X;
    K = (N - 1) * (N - 2) / 2;
    if(X > K){ cout << -1; return 0; }
    K -= X;
    memset(Use, true, sizeof Use);
    for(int i=1; i<=N-2; i++){
        if(K >= N-i-1) K -= N-i-1, Use[i] = false;
    }
    cout << N << " ";
    for(int i=1; i<=N-2; i++) if(Use[i]) cout << i << " ";
    cout << N - 1 << " ";
    for(int i=N-2; i>=1; i--) if(!Use[i]) cout << i << " ";
```

```
}
```

BOJ 14464 소가 길을 건너간 이유 4

회의실 배정과 비슷한 논리로, 닭을 끝점 기준으로 정렬하자.

닭을 차례대로 보면서, 도와줄 수 있는 T_i가 가장 작은 소를 도와주는 그리디가 성립한다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;

int C, N;
PII A[20202];
multiset<int> S;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> C >> N;
    for(int i=1; i<=C; i++){
        int t; cin >> t; S.insert(t);
    }
    for(int i=1; i<=N; i++){
        cin >> A[i].x >> A[i].y;
    }
    sort(A+1, A+N+1, [](PII p1, PII p2){
        if(p1.y != p2.y) return p1.y < p2.y;
        return p1.x < p2.x;
    });

    int ans = 0;
    for(int i=1; i<=N; i++){
        auto it = S.lower_bound(A[i].x);
        if(it != S.end() && *it <= A[i].y) ans++, S.erase(it);
    }
    cout << ans;
}
```

BOJ 14908 구두 수선공

강의 슬라이드 참고

```
#include <bits/stdc++.h>
using namespace std;

struct Task{
    int idx, deadline, penalty;
    bool operator < (const Task &task) const {
        int t1 = deadline * task.penalty;
        int t2 = task.deadline * penalty;
        if(t1 != t2) return t1 < t2;
        return idx < task.idx;
    }
};
```

```
int N;
Task A[1010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].deadline >> A[i].penalty, A[i].idx = i;
    sort(A+1, A+N+1);
    for(int i=1; i<=N; i++) cout << A[i].idx << " ";
}
```

BOJ 2878 캔디캔디

$x^2 - (x-1)^2 \geq (x-1)^2 - (x-2)^2$이므로 가장 큰 것을 1씩 깍아주는 것이 최적이다. 이 연산을 20억 번 하면 당연히 시간 초과이므로 최적화를 해야 한다.

모든 값을 x 이하로 만들 수 있는가?라는 결정 문제를 이용해 파라메트릭 서치를 수행하자. 가능한 x의 최솟값을 찾아서 모든 수를 x 이하로 만든 뒤, 남은 연산 횟수는 가장 큰 것을 깍아주면 된다.

가장 큰 것을 깍는 연산은 최대 $N-1$번만 하면 되고(N 이상이면 x를 1만큼 더 감소시킬 수 있음), 이 작업은 우선순위 큐를 이용해 효율적으로 할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using ull = unsigned long long;

ll M, N, A[101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> M >> N;
    for(int i=1; i<=N; i++) cin >> A[i];

    ll l = 0, r = 2e9 + 10;
    while(l < r){
        ll m = l + r >> 1;
        ll require = accumulate(A+1, A+N+1, 0LL, [&m](ll a, ll b){ return a +
max(0LL, b - m); });
        if(require <= M) r = m;
        else l = m + 1;
    }
    for(int i=1; i<=N; i++){
        M -= max(0LL, A[i] - r);
        A[i] = min(A[i], r);
    }

    priority_queue<ll> pq;
    for(int i=1; i<=N; i++) if(A[i]) pq.push(A[i]);
    for(int i=0; i<M && !pq.empty(); i++){
        ll now = pq.top(); pq.pop();
        if(now - 1 > 0) pq.push(now - 1);
    }

    ull ans = 0;
    while(pq.size()){
```

```
        ull now = pq.top(); pq.pop();
        ans += now * now;
    }
    cout << ans;
}
```

BOJ 2180 소방서의 고민

강의 슬라이드 참고

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using PLL = pair<ll, ll>;
constexpr ll MOD = 40'000;

int N;
PLL A[202020];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    sort(A+1, A+N+1, [](PLL p1, PLL p2){
        if(p1 == PLL(0,0)) return false;
        if(p2 == PLL(0,0)) return true;
        ll p12 = p1.y * p2.x, p21 = p2.y * p1.x;
        return p12 < p21;
    });

    ll ans = 0, ti = 0;
    for(int i=1; i<=N; i++){
        ans = (ans + A[i].x * ti + A[i].y) % MOD;
        ti = ans;
    }
    cout << ans;
}
```

BOJ 18921 Cost Of Subtree

Kruskal's Algorithm와 비슷하게, 가중치가 큰 간선부터 넣어보면서 답을 구해야 한다.

```
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
using namespace std;
using ll = long long;

struct Edge{
    ll s, e, x;
    bool operator < (const Edge &edge) const { return x < edge.x; }
    bool operator > (const Edge &edge) const { return x > edge.x; }
};

int N, P[101010], S[101010];
```

```
vector<Edge> edges;

int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Merge(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return false;
    P[u] = v; S[v] += S[u];
    return true;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; edges.resize(N - 1);
    for(auto &i : edges) cin >> i.s >> i.e >> i.x;
    sort(all(edges), greater<>());

    iota(P+1, P+N+1, 1);
    fill(S+1, S+N+1, 1);
    ll mx = 0;
    for(auto i : edges){
        if(Merge(i.s, i.e)) mx = max(mx, i.x * (S[Find(i.s)] - 1));
    }
    cout << mx;
}
```

BOJ 21761 초직사각형

강의 슬라이드 참고

```
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
using namespace std;
using ll = long long;
using LL = __int128_t;

struct Card{
    int id, idx;
    Card() = default;
    Card(int id, int idx) : id(id), idx(idx) {}
};

int N, K, A[4];
vector<ll> V[4], S[4];

bool operator > (const Card &a, const Card &b){
    LL a1 = S[a.id][a.idx], a2 = S[a.id][a.idx-1];
    LL b1 = S[b.id][b.idx], b2 = S[b.id][b.idx-1];
    return a1*b2 > b1*a2; // a1/a2 > b1/b2
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=0; i<4; i++) cin >> A[i];
    for(int i=0; i<N; i++){
        char c; int x; cin >> c >> x;
        V[c-'A'].push_back(x);
```

```
    }
    for(int i=0; i<4; i++){
        sort(all(V[i]), greater<>());
        V[i].insert(V[i].begin(), A[i]);
        partial_sum(all(V[i]), back_inserter(S[i]));
    }

    vector<Card> cards;
    for(int i=0; i<4; i++){
        for(int j=1; j<V[i].size(); j++) cards.emplace_back(i, j);
    }
    sort(all(cards), greater<>());
    cards.resize(K);
    for(auto [id,idx] : cards) cout << char('A'+id) << " " << V[id][idx] <<
"\n";
}
```

BOJ 1422 숫자의 신

여러 번 사용하는 $K-N$개의 수는 N개의 수들 중 가장 큰 수를 $K-N$번 사용하면 된다.

K개의 수를 나열한 결과가 최대가 되도록 하는 배치 순서는 exchange argument를 생각해보면 쉽게 알 수 있다.

```
#include <bits/stdc++.h>
using namespace std;

int N, K;
string A[1010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K; for(int i=1; i<=N; i++) cin >> A[i];
    string largest = *max_element(A+1, A+N+1, [](const string &s1, const string
&s2){
        if(s1.size() != s2.size()) return s1.size() < s2.size();
        return s1 < s2;
    });
    while(N < K) A[++N] = largest;
    sort(A+1, A+N+1, [](const string &s1, const string &s2){
        return s1 + s2 > s2 + s1;
    });
    for(int i=1; i<=N; i++) cout << A[i];
}
```

BOJ 21740 도도의 수학놀이

BOJ 1422 숫자의 신과 비슷하게 하면 된다.

```
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
using namespace std;

int N;
string A[1010101], R[1010101];
```

```
string Reverse(const string &s){
    string ret = s; reverse(all(ret));
    for(auto &i : ret){
        if(i == '6') i = '9';
        else if(i == '9') i = '6';
    }
    return ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i], R[i] = Reverse(A[i]);
    int largest = max_element(R+1, R+N+1, [](const string &s1, const string &s2)
{
        if(s1.size() != s2.size()) return s1.size() < s2.size();
        return s1 < s2;
    }) - R;

    vector<int> O(N+1);
    iota(all(O), 1); O.back() = largest;
    sort(all(O), [](int s1, int s2){
        return R[s1] + R[s2] > R[s2] + R[s1];
    });
    reverse(all(O));
    for(auto i : O) cout << A[i];
}
```

BOJ 12776 Swap Space

문제를 간단하게 요약하자면...

> N개의 순서쌍 (A_i, B_i)가 주어진다.
>
> x는 0부터 시작해서 N개의 순서쌍을 한 번씩 방문한다. 순서쌍 (A_i, B_i)를 방문하면 x는 A_i만큼 감소한 다음 B_i만큼 증가한다.
>
> N개의 순서쌍을 모두 방문하는 동안 x의 최솟값을 최대화시키는 문제다. (절댓값을 출력)

$V_i = B_i - A_i$라고 정의하자. $V_x \geq 0 \wedge V_y < 0$이면 x를 먼저 선택하는 것이 무조건 이득이다. 둘 다 0 이상이거나 0 미만인 경우를 잘 생각해보자.

- $V_x, Y_y \geq 0$인 경우 $A_x < A_y$가 되도록 정렬
  - A_x가 먼저 오도록 정렬했을 때의 최솟값은 $\min\{-A_x, -A_x + B_x - A_y\}$
  - A_y가 먼저 오도록 정렬했을 때의 최솟값은 $\min\{-A_y, -A_y + B_y - A_x\}$
  - $-A_y < A_x$이고 $-A_y < -A_x + B_x - A_y$이므로 A_x가 먼저 오도록 정렬하는 것이 이득이다.
- $V_x, V_y < 0$인 경우 $B_x > B_y$가 되도록 정렬
  - B_x가 먼저 오도록 정렬했을 때의 최솟값은 $\min\{-A_x, -A_x + B_x - A_y\}$
  - B_y가 먼저 오도록 정렬했을 때의 최솟값은 $\min\{-A_y, -A_y + B_y - A_x\}$
  - $-A_y + B_y - A_x < -A_x$이고 $-A_y + B_y - A_x < -A_x + B_x - A_y$이므로 B_x가 먼저 오도록 정렬하는 것이 이득이다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
```

```
struct Info{
    ll a, b, idx;
    Info() : a(0), b(0), idx(0) {}
    bool operator < (const Info &t) const {
        ll le = b - a, ri = t.b - t.a;
        if(le >= 0 && ri < 0) return true;
        if(le < 0 && ri >= 0) return false;
        if(le < 0 && ri < 0) return b > t.b;
        if(le >= 0 && ri >= 0) return a < t.a;
        return idx < t.idx;
    }
    Info& operator += (const Info &t){
        ll down = min(-a, -a + b - t.a);
        ll final = -a + b - t.a + t.b;
        this->a = -down;
        this->b = final - down;
        return *this;
    }
};

int N;
Info A[1010101];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].a >> A[i].b, A[i].idx = i;
    sort(A+1, A+N+1);

    Info res;
    for(int i=1; i<=N; i++) res += A[i];
    cout << res.a;
}
```

BOJ 5530 JOIOI 탑

미니 JOIOI 탑을 x개 이상 만들 수 있는지 판단하는 결정 문제를 이용해 파라메트릭 서치를 하면 문제를 해결할 수 있습니다.

미니 JOIOI 탑을 x개 만들 수 있다면, 가장 뒤에 있는 `I` x개가 `JOI, IOI`의 끝 문자가 되도록 하는 해가 존재합니다. 그러므로 결정 문제를 해결할 때 가장 뒤에 있는 `I` x개를 끝 문자로 고정시킨 뒤, 그리디하게 `JOI, IOI`를 만들면 됩니다.

```
#include <bits/stdc++.h>
using namespace std;

int N;
char A[1010101];
vector<int> Pos;

bool Check(int make){
    int a = 0, b = 0, c = 0, sep = Pos[Pos.size() - make];
    for(int i=1; i<=N; i++){
        if(A[i] == 'J') a++;
        if(A[i] == 'O'){
            if(a > 0) a--, b++;
```

```
        }
        if(A[i] == 'I'){
            if(i < sep) a++;
            else if(b > 0) b--, c++;
        }
    }
    return make == c;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> (A+1);
    for(int i=1; i<=N; i++) if(A[i] == 'I') Pos.push_back(i);
    if(Pos.empty() || !Check(1)){ cout << 0; return 0; }

    int l = 1, r = Pos.size();
    while(l < r){
        int m = l + r + 1 >> 1;
        if(Check(m)) l = m;
        else r = m - 1;
    }
    cout << l;
}
```

BOJ 18596 Monster Hunter

BOJ 12776 Swap Space와 동일한 상황인데, Random Access가 가능한 배열이 아닌, 부모 정점을 방문한 적이 있어야 자식 정점을 방문할 수 있는 Rooted Tree라는 것이 문제다.

아직 방문하지 않은 정점 중에서, Random Access가 가능하다면 가장 먼저 방문할 정점 v을 선택하자. 만약 v의 부모 정점을 방문했다면, 바로 다음에 v번 정점을 방문하는 것이 최적일 것이다. 그러므로 v의 부모 정점과 v번 정점을 하나로 합쳐줄 수 있다.

정점을 합칠 때마다 정점의 개수가 1씩 감소하므로, 이 과정을 $N-1$번 반복하면 하나의 정점만 남게 된다.

정점을 합치는 것은 Union-Find로, Random Access가 가능할 때 가장 먼저 방문할 정점을 구하는 것은 우선순위 큐로 처리할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

struct Info{
    ll a, b, idx;
    Info() : Info(0, 0, 0) {}
    Info(ll a, ll b, ll idx) : a(a), b(b), idx(idx) {}
    bool operator < (const Info &t) const {
        ll le = b - a, ri = t.b - t.a;
        if(le >= 0 && ri < 0) return false;
        if(le < 0 && ri >= 0) return true;
        if(le < 0 && ri < 0) return b < t.b;
        if(le >= 0 && ri >= 0) return a > t.a;
        return idx < t.idx;
    }
    Info& operator += (const Info &t){
        ll down = min(-a, -a + b - t.a);
```

```
        ll final = -a + b - t.a + t.b;
        this->a = -down;
        this->b = final - down;
        return *this;
    }
};

int N, UF[101010], P[101010], C[101010];
Info A[101010];
vector<int> G[101010];
priority_queue<Info> pq;

void Clear(){
    iota(UF+1, UF+N+1, 1);
    memset(C+1, 0, sizeof(C[0]) * N);
    for(int i=1; i<=N; i++) G[i].clear();
}

int Find(int v){ return v == UF[v] ? v : UF[v] = Find(UF[v]); }

void DFS(int v, int b=-1){
    for(auto i : G[v]) if(i != b) P[i] = v, DFS(i, v);
}

void Solve(){
    cin >> N; Clear();
    A[1] = {0, 0, 1};
    for(int i=2; i<=N; i++){
        cin >> A[i].a >> A[i].b; A[i].idx = i;
        pq.push(A[i]);
    }
    for(int i=1; i<N; i++){
        int s, e; cin >> s >> e;
        G[s].push_back(e); G[e].push_back(s);
    }
    DFS(1);

    while(pq.size()){
        auto t = pq.top(); pq.pop();
        int now = t.idx, nxt = Find(P[now]);
        if(C[now]) continue; C[now] = 1;
        A[nxt] += A[now]; UF[now] = nxt;
        if(nxt != 1) pq.push(A[nxt]);
    }
    cout << A[1].a << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int T; cin >> T;
    for(int tc=1; tc<=T; tc++) Solve();
}
```